

Inhalt

- Grundbegriffe
- Sichtbarkeit von Namen und Bindung
- Speicherorganisation für Prozeduren
- Adressierung von Variablen
 - Globale Variablen
 - Prozeduraufruf
- Berechnung der Adressumgebungen
- Prozedureintritt und Prozedurverlassen
 - Prozedurdeklaration
 - Prozeduraufruf
 - Parameterübergabe
 - Zugriff auf Variablen und formale Parameter
- Formale Prozeduren
- Hauptprogramm

Lernziele

- Die prinzipiellen Probleme, die durch sich dynamisch ändernde Strukturierungsmöglichkeiten (z.B. Prozeduren) entstehen, kennen und erklären können, wie man sie prinzipiell löst; insbesondere wie man
 - Variable adressiert
 - Die jeweilige Adressumgebung berechnet
 - Das Zusammenspiel zwischen Prozedurdeklaration und -aufruf regelt
 - Verschiedene Parameterübergabemechanismen umsetzt
- Beschreiben können, wie man das Problem von Prozeduren als Parameter löst

Ziel der folgenden Überlegungen

- Codefunktionen für Prozedurdeklaration und -aufruf
 - Funktionen (als Spezialfall) mitbehandelt
 - Wichtigster Aspekt: Zusammenspiel zwischen aufrufender und aufgerufener Prozedur

Zu unterscheiden (bei Prozeduren und Funktionen)

- **Definierendes Vorkommen** = Deklaration mit
 - Namen der Prozedur (oder Funktion)
 - Spezifikation (= Namen und Typen) der formalen Parameter
 - (Angabe des Ergebnistyps bei Funktionen)
 - (lokale) Deklarationen (für Variable, Funktionen oder Prozeduren)
 - Anweisungsteil („Rumpf“)
- **Angewandtes Vorkommen**
 - Aufruf (mit aktuellen Parametern)

Anders als etwa bei C können auch Prozeduren lokal (zu anderen Prozeduren) deklariert werden. Damit: beliebige Schachtelung von Prozeduren möglich

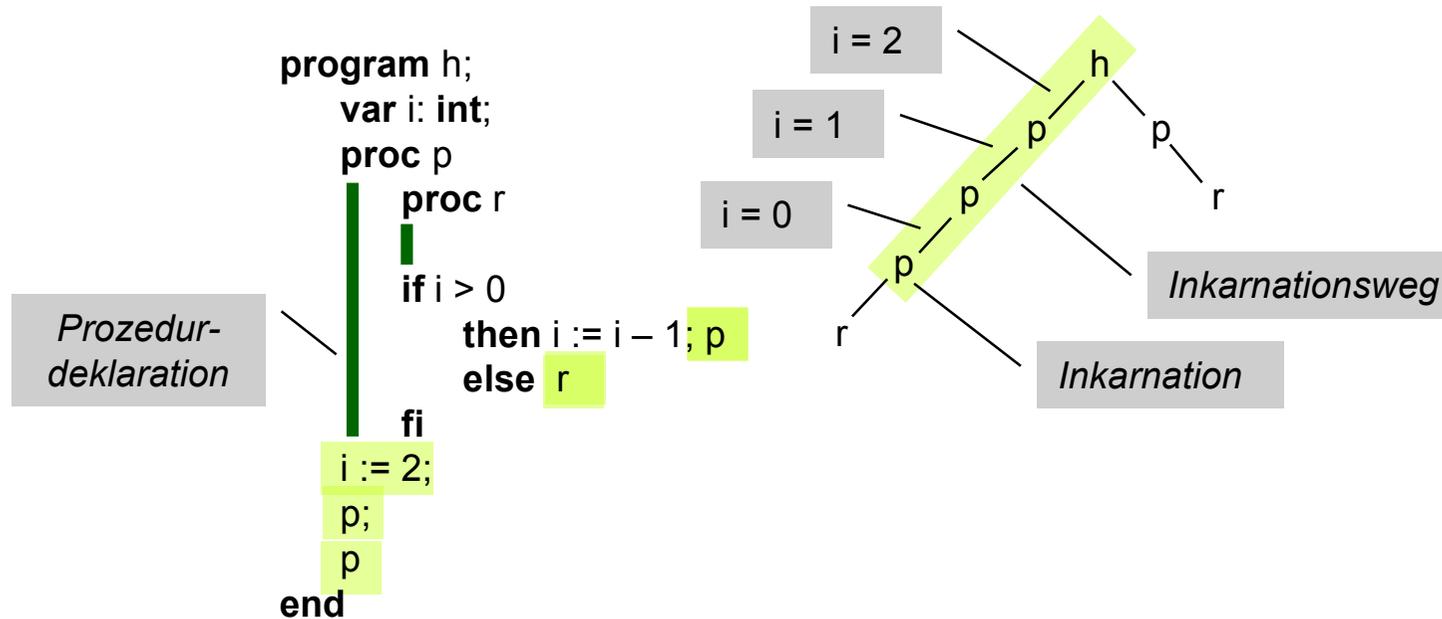
Zu beachten (bei Prozeduren)

- Aufgerufene (noch nicht beendete) Prozedur kann ihrerseits andere Prozedur (oder sich selbst) aufrufen

Begriffe (zur Beschreibung dieses Phänomens)

- **Aufrufbaum** (eines Programmlaufs)
Geordneter Baum, der bei Ausführung eines Programms entstehenden Prozeduraufrufe („Aufrufstruktur“)
- **Inkarnation** (einer Prozedur p)
Vorkommen von p im Aufrufbaum *Können auch mehrere sein*
- **Inkarnationsweg** (einer Inkarnation) *„Aufruf-Geschichte“*
Weg von der Wurzel des Aufrufbaums zu dieser Inkarnation
- **Lebendige Inkarnation**
Zugehörige Prozedur ist aufgerufen, aber nicht beendet *Ist von der betrachteten Programmstelle abhängig*

Beispiel (Programm und Aufrufbaum)



Beachte

- Mehr als ein Aufrufbaum für ein Programm (abhängig von Eingabe / Wertebelegung)

Besonderheit im obigen Beispiel:
genau ein endlicher Aufrufbaum

- Unendliche Aufrufbäume

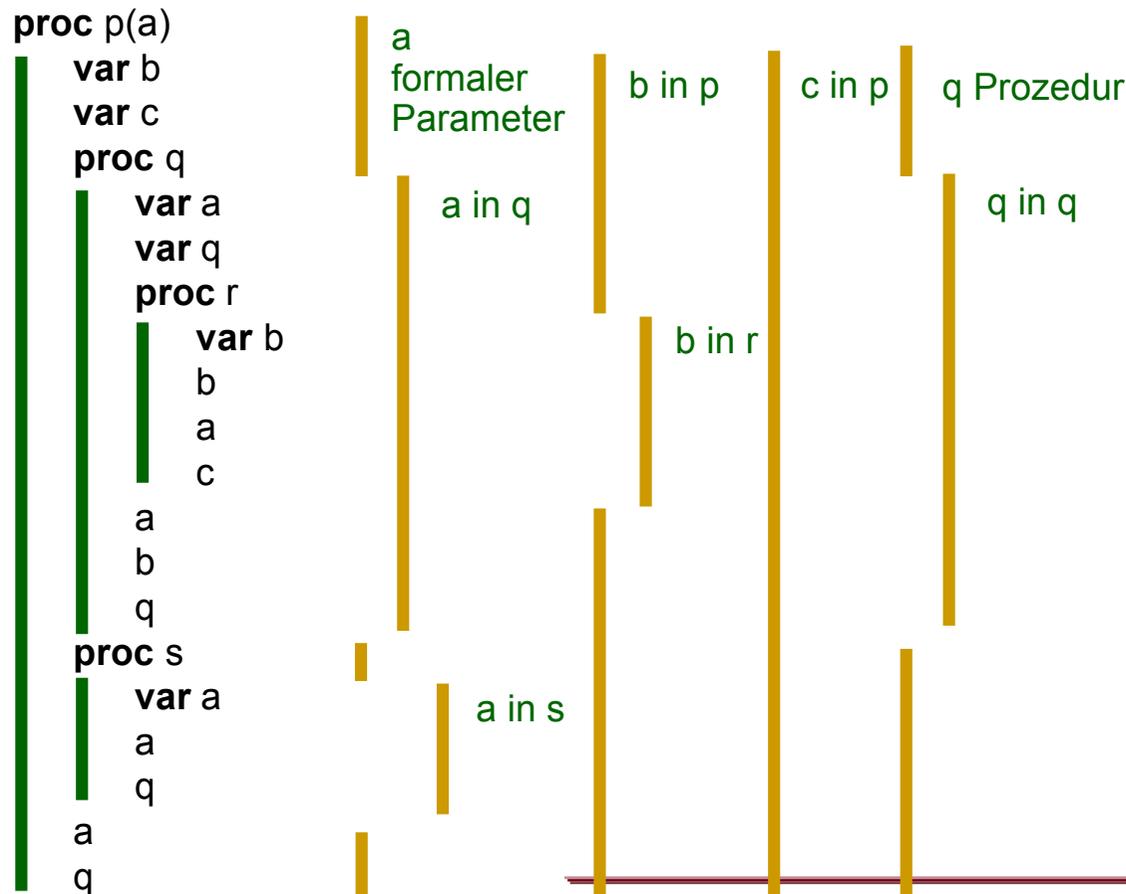
z.B. bei Nicht-Terminierung

Sichtbarkeitsregel (in Algol-ähnlichen Sprachen)

*Fast alle imperativen Sprachen
Ausnahme: Fortran*

- Definierendes Vorkommen eines Namens ist sichtbar in der Prozedur (Block), deren Deklarations-/Spezifikationsteil die Definition enthält, abzüglich aller echt umfassten Prozeduren (Blöcke), die eine Neudefinition enthalten

Sichtbarkeitsbereiche



Vorgehensweise
(für Namen x)

- Def. Vorkommen von x suchen
- Umfassenden Block bestimmen
- Innere Blöcke mit def. Vorkommen von x entfernen

Verschiedene Möglichkeiten (der Zuordnung zwischen def. und angew. Vorkommen)

• Statische Bindung

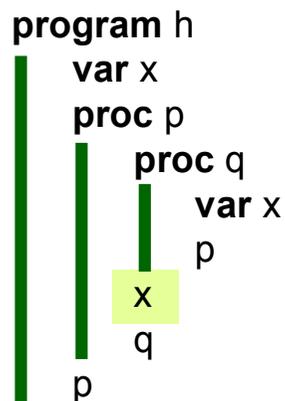
- Zuordnung gemäß obiger Sichtbarkeitsregel (statisch = nur von Programmtext abhängig): angewandtes Vorkommen (bei Ausführung) korrespondiert mit zuletzt kreierter Inkarnation des statisch zugeordneten definierenden Vorkommens

• Dynamische Bindung

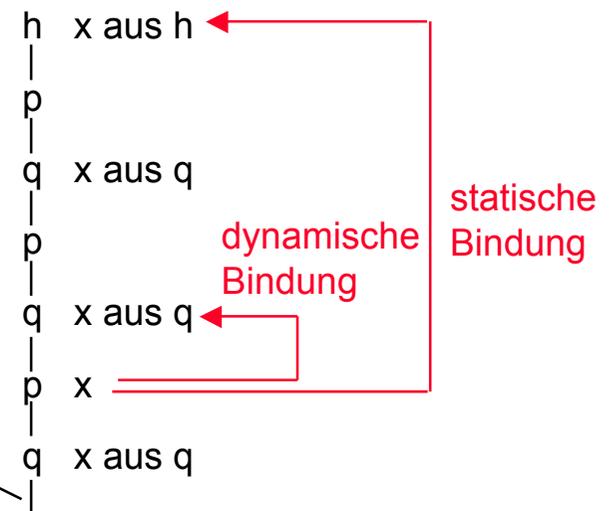
- Angewandtes Vorkommen (bei Ausführung) korrespondiert mit zuletzt kreierter Inkarnation (unabhängig davon, wo definierendes Vorkommen auftritt)

Beispiel (statische und dynamische Bindung anhand des Aufrufbaums)

Beispiel-Programm



Aufrufbaum

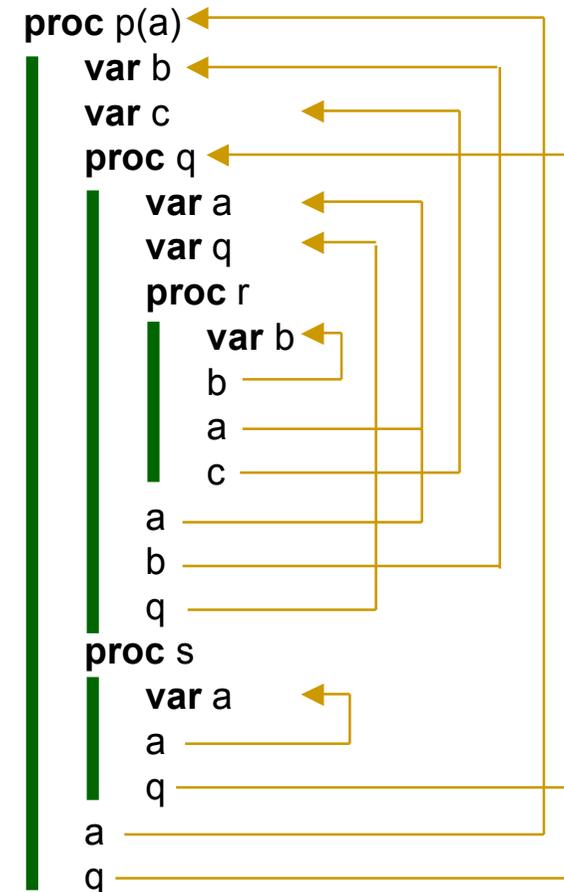


Unendlicher Aufrufbaum

Ermittlung eines definierenden Vorkommens (über Inkarnationsweg)

- *Dynamische Bindung*
Erstes definierendes Vorkommen im Aufrufbaum auf dem Rückweg zur Wurzel
- *Statische Bindung*
Definierendes Vorkommen in letzter Inkarnation der innersten umfassenden Prozedur
- **Fazit:** Aufrufbaum ungeeignet zum effizienten Auffinden der „richtigen“ Inkarnation bei statischer Bindung
⇒ anderes Konzept erforderlich (s.u.)

Beispiel (statische Bindung)



Statischer Vorgänger (einer Prozedur p)

- Letzte Inkarnation einer p direkt umfassenden Prozedur

Baum der statischen Vorgänger (zu einem Inkarnationsweg)

- Knoten: Inkarnationen
- p ist Sohn von q: q ist letzte Inkarnation der p direkt umfassenden Prozedur

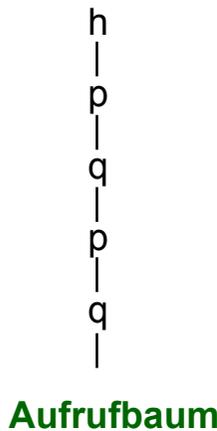
Damit (im Baum der statischen Vorgänger)

- „richtige“ Inkarnationen (globaler Variablen) eines angewandten Vorkommens auf Weg von zugehöriger Inkarnation zur Wurzel

Beispiele

```

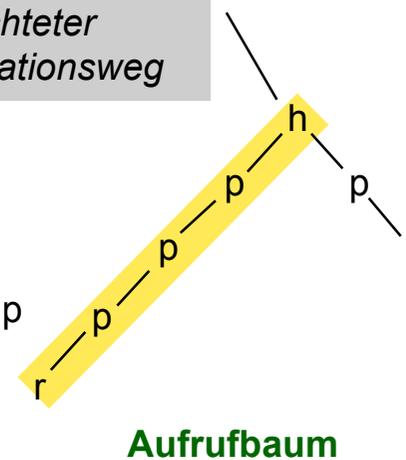
program h
  var x
  proc p
    proc q
      var x
      p
    end
  end
end
  
```



```

program h;
  var i: int;
  proc p
    proc r
      if i > 0
        then i := i - 1; p
        else r
      fi
    end
  end
  i := 2; p; p
end
  
```

Betrachteter Inkarnationsweg



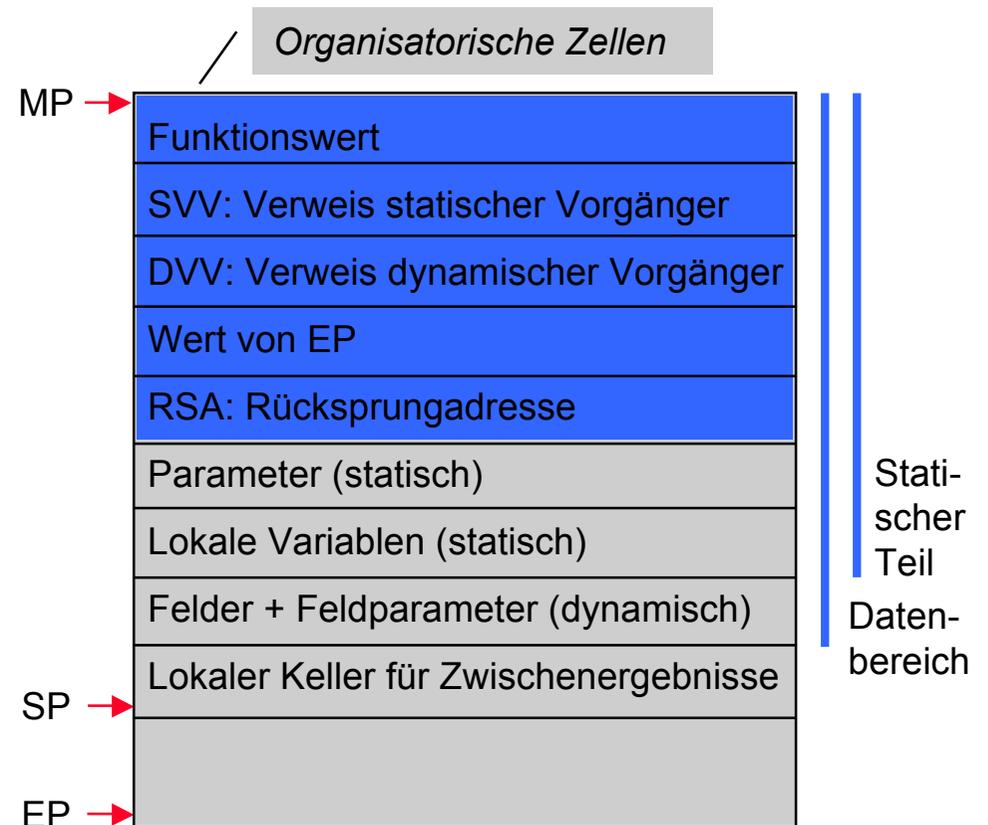
Laufzeitkeller

- Gleichzeitige Realisierung zweier Strukturen
 - *Inkarnationsweg*: als Folge von konsekutiven Speicherbereichen für lebende Inkarnationen
 - *Statischer Vorgängerbaum* (zum Inkarnationsweg): durch Zeiger

Für jede Inkarnation einer Prozedur

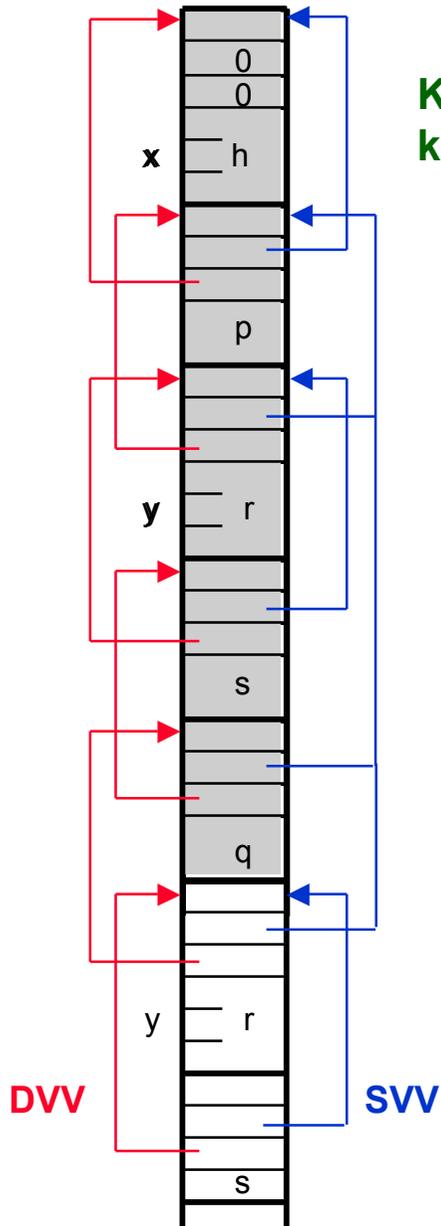
Kellerrahmen, bestehend aus

- Datenbereich (der Inkarnation)
 - Statischer Teil
 - Organisatorische Zellen, je eine für
 - Ergebnis (für Funktionen)
 - Verweis auf statischen Vorgänger
 - Verweis auf dynamischen Vorgänger
 - Stand von EP (s.u.)
 - Rücksprungadresse
 - Statische Parameter (inkl. Deskriptoren)
 - (stat.) lokale Variablen (inkl. Deskriptoren)
 - Dynamischer Teil (Felder + Feldparameter)
- Lokaler Keller
(für die Auswertung des Rumpfes)



Verwendete Register

- MP („mark pointer“)
Zeiger auf Anfang des Kellerrahmens der aktuellen Inkarnation
 - SP („stack pointer“)
Zeiger auf „oberste“ belegte Zelle des lokalen Kellers
 - EP („extreme stack pointer“)
Zeiger auf „höchste“ (während Ausführung der Prozedur) belegte Kellerzelle
(entspricht maximalem Platzbedarf zur Auswertung des Prozedurrumpfes)
- Neu: Basis für die relative Adressierung innerhalb einer Prozedur*
- wie gehabt*



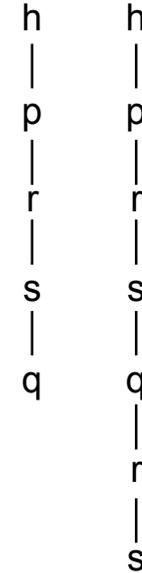
**Keller-
konfiguration**

Programm

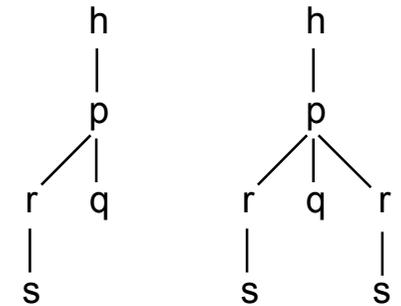
```

program h
  var x
  proc p
    :
    proc q
      :
      r
      :
      proc r
        var y
        proc s
          x + y
          q
          s
        r
      p
  
```

**Aufrufbaum /
Inkarnationsweg**



**Baum der statischen
Vorgänger**



Offensichtlich

- Wegen geschachtelter Sichtbarkeitsbereiche und dynamisch kreierter neuer Inkarnationen keine statischen, absoluten Adressen zuordenbar

Stattdessen

- Lokale Variablen
 - Statische Relativadresse, relativ zum Anfang des Kellerrahmens
 - Zugriff über MP-Register
- Globale Variablen
 - Zugriff über SV-Verweis
 - Problem
 - Mehrfach-Verfolgen von SV-Verweisen
 - Frage: wie oft?

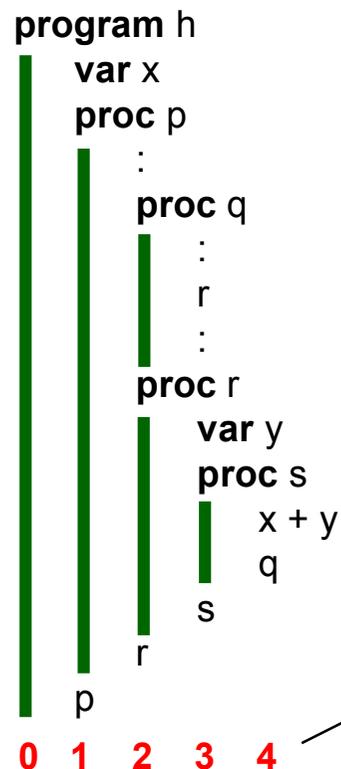
Zur Adressierung globaler Variablen erforderlich

- Schachtelungstiefe (eines Programmkonstrukts)

Schachtelungstiefe (eines Programmkonstrukts)

- Hauptprogramm: 0
- Definierendes (angewandtes) Vorkommen eines Namens im Deklarations-(Anweisungs-)teil einer Einheit mit Schachtelungstiefe n : $n + 1$

Beispiel



Schachtelungstiefen

Definierende Vorkommen		Angewandte Vorkommen	
p	1	p	1
q	2	q	4
r	2	r (in p)	2
		r (in q)	3
s	3	s	3
x	1	x	4
y	3	y	4

Invariante (ISV)

- Zu jedem Zeitpunkt der Ausführung eines Programms gilt
 - In jedem auf dem Keller angelegten Rahmen für eine Inkarnation einer Prozedur p zeigt der SV-Verweis auf den Kellerrahmen der „richtigen“ Inkarnation der p direkt umfassenden Einheit, d.h.
 - In Programmen ohne formale Prozeduren:
jeweils jüngste noch lebende Inkarnation der p direkt umfassenden Programmeinheit
 - In Programmen mit formalen Prozeduren
s.u.

Vorgehensweise

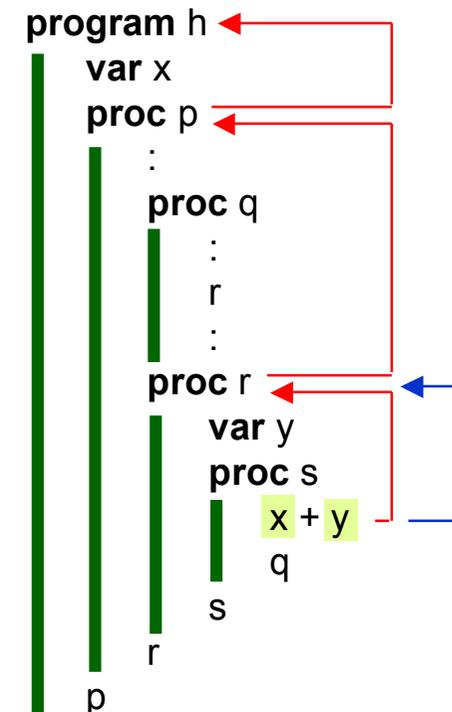
- Zugriff auf globale Variablen: Adressberechnung mit ISV
- Prozeduraufruf: Sicherstellen von ISV für neu anzulegenden Rahmen

Zugriff auf globale Variablen

- Voraussetzung: ISV gilt
- Zugriff auf globale Namen (Variablen und Prozeduren)
 - Situation
 - Angewandtes Vorkommen auf ST n ,
 - Definierendes Vorkommen auf ST m ($m \leq n$)
 - $(n-m)$ -maliges Verfolgen des SV-Verweises ergibt AA des gesuchten Rahmens

Beispiele

- Zugriff auf x in s
 - ST dieses (angew.) Vorkommens: 4
 - ST des zugehörigen def. Vorkommens in h : 1
 - \Rightarrow 3-maliges Verfolgen des SV-Verweises ergibt Anfang des zugehörigen Kellerrahmens (von h)
- Zugriff auf y in s
 - ST dieses (angew.) Vorkommens: 4
 - ST des zugehörigen def. Vorkommens in r : 3
 - \Rightarrow 1-maliges Verfolgen des SV-Verweises ergibt Anfang des zugehörigen Kellerrahmens (der letzten Inkarnation von r)



Neue P-Befehle (zum Zugriff über Schachtelungstiefe)

Befehl	Bedeutung	Kommentar
lod T p q	SP := SP+1; STORE[SP] := STORE[<u>base(p, MP) + q</u>]	p Differenz der Schachtelungstiefen q Relativadresse
lda p q	SP := SP+1; STORE[SP] := <u>base(p, MP) + q</u>	
str T p q	STORE[<u>base(p, MP) + q</u>] := STORE[SP] SP := SP-1;	

entspricht
ldo
ldc a
sro

Unterschied:

Dabei

• $\text{base}(p, a) =_{\text{def}} \text{if } p = 0 \text{ then } a \text{ else } \text{base}(p-1, \text{STORE}[a + 1]) \text{ fi}$

Differenz der ST

Aktueller MP

AA des Keller-rahmens der Prozedur, wo die Deklaration zu finden ist

entspricht lokaler Variable

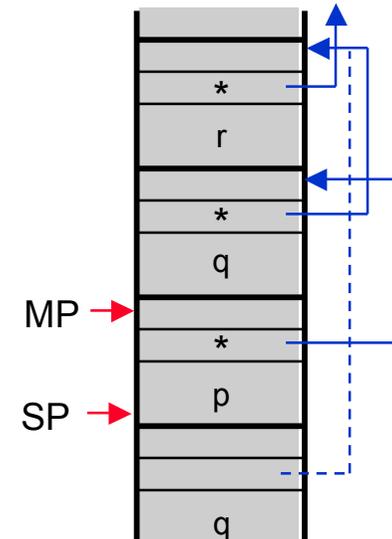
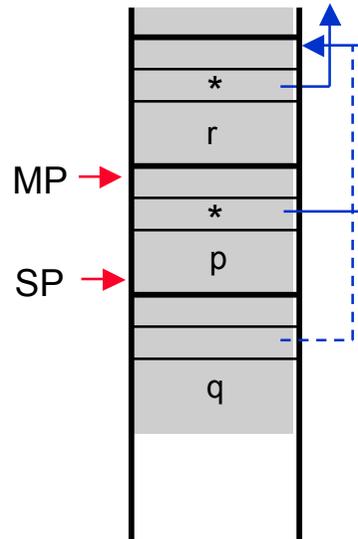
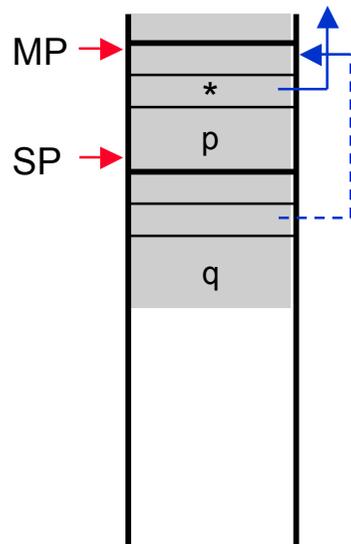
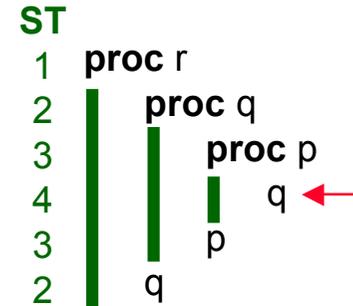
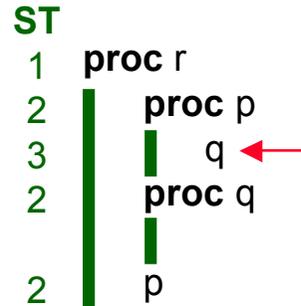
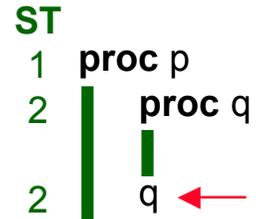
SV-Verweis

Prozeduraufruf

- Voraussetzung: ISV gilt
- Sicherstellen von ISV bei Abarbeitung Prozeduraufruf (d.h. Besetzung SV-Verweis in anzulegendem Kellerrahmen)
 - Situation
 - Aufruf von q in p auf ST n
 - Definierendes Vorkommen von q auf ST m ($m \leq n$)
 - Statischer Vorgänger von q durch $(n - m)$ -maliges Verfolgen des SV-Verweises in p
 - $m = n$: angewandtes und definierendes Vorkommen von q auf gleicher Stufe (d.h. in p)
 - ⇒ statischer Vorgänger von q = p
 - $m < n$: statischer Vorgänger von q auch (direkter oder indirekter) Vorgänger von p
 - ⇒ statischer Vorgänger ergibt sich durch $(n-m)$ -maliges Verfolgen der SV-Kette, die in p beginnt



Aufruf- und Kellersituationen (Aufrufer: p; Aufgerufener: q)



Erweiterung des bisherigen Konzepts

- Adressen und Schachtelungstiefen
- Berücksichtigung der Sichtbarkeitsregel

Adressen und Schachtelungstiefen

- Bindung aller definierenden Vorkommen von Namen
 - Variablennamen: an Relativadresse + Schachtelungstiefe
 - Prozedurnamen: an symbolische Marke + Schachtelungstiefe
- Somit
 - $\text{Adr_Umg} = \text{Id} \rightarrow \text{Adr} \times \text{ST}$
 - wobei Adr
 - Für Variablennamen: Relativadresse in Kellerrahmen
 - Für Prozedurnamen: (absolute) Adresse in CODE

Berücksichtigung der Sichtbarkeitsregel

- Verarbeitung des Deklarationsteils einer Prozedur von außen nach innen
- Lokale Adressumgebung mit „Überschreiben“ nicht sichtbarer Namen
(Vorteil der rekursiven Definition der code-Funktion: nach Abarbeitung der Prozedurdeklaration liegt wieder „alte“ Adressumgebung vor)

Verarbeitung von Parameterspezifikationen (= Veränderung der Adressumgebung)

- $\text{elab_specs}: \text{Spec}^* \times \text{Adr_Umg} \times \text{Adr} \times \text{ST} \rightarrow \text{Adr_Umg} \times \text{Adr}$
 - 1. „freie“ Adresse
 - nächste „freie“ Adresse
- mit
 - $\rho[(n_a, st) / x]$ identisch mit ρ bis auf Wert n_a für x
- hat man
 - $\text{elab_specs} () \rho n_a st =_{\text{def}} (\rho, n_a)$
 - $\text{elab_specs} (\text{var } x: t; \text{specs}) \rho n_a st =_{\text{def}} \text{elab_specs specs } \rho[(n_a, st) / x] (n_a + 1) st$
 - $\text{elab_specs} (\text{value } x: t; \text{specs}) \rho n_a st =_{\text{def}} \text{elab_specs specs } \rho[(n_a, st) / x] (n_a + \text{gr}(t)) st$ für statische Typen t
 - $\text{elab_specs} (\text{value } x: \text{array } [u_1..o_1, \dots, u_k..o_k] \text{ of } t; \text{specs}) \rho n_a st =_{\text{def}} \text{elab_specs specs } \rho' (n_a + 3k + 2) st$
 mit $\rho' = \rho[(n_a, st) / x][[(n_a + 2i + 1, st) / u_i]_{i=1..k} [(n_a + 2i + 2, st) / o_i]_{i=1..k}]$

Platzbedarf

- Variable: 1 Speicherplatz
- (value-) Feld: $3k+2$ Speicherplätze für Deskriptor
- (sonstiger) value-Parameter: Speicherplatz entsprechender Größe

Beispiel

- Geg.: **var x, f, k: integer; z: boolean;**
proc p (value k: integer; var x: boolean; value f: array [m..n, s..t] of char)
- Annahme: st = 1
- Verarbeitung der Parameterspezifikationen
 - $\rho = \{(x \rightarrow (5,1), f \rightarrow (6,1), k \rightarrow (7,1), z \rightarrow (8,1))\}$, $n_a = 9, st = 1$
 - $\rho = \{(x \rightarrow (5,1), f \rightarrow (6,1), k \rightarrow (7,1), z \rightarrow (8,1), p \rightarrow (l, 1))\}$,
 $n_a = 5, st = 2$ — *n_a wird bei Verarbeitung einer Prozedurdeklaration zurückgesetzt, st erhöht*
 - $\rho = \{(x \rightarrow (6,2), f \rightarrow (6,1), k \rightarrow (5,2), z \rightarrow (8,1), p \rightarrow (l, 1))\}$, $n_a = 7, st = 2$
 - $\rho = \{(x \rightarrow (6,2), f \rightarrow (7,2), k \rightarrow (5,2), z \rightarrow (8, 1), p \rightarrow (l, 1),$
 $(m \rightarrow (10,2), n \rightarrow (11,2), s \rightarrow (12,2), t \rightarrow (13,2))\}$, $n_a = 15, st = 2$
- Auf Relativadressen 7 bis 14: Deskriptor für f
- „fehlende“ Einträge für Deskriptor
 - (8,2): Feldgröße
 - (9,2): Subtrahend für fiktive AA
 - (14,2): Spanne d_2

Prozedurbezeichner p zugeordnete symbolische Marke (Details, s.u.)

Verarbeitung der Deklarationen lokaler Variablen

- $\text{elab_vdecls}: \text{Vdecl}^* \times \text{Adr_Umg} \times \text{Adr} \times \text{ST} \rightarrow \text{Adr_Umg} \times \text{Adr}$
- mit

- $\text{elab_vdecls} () \rho \ n_a \ st =_{\text{def}} (\rho, \ n_a)$

- $\text{elab_vdecls} (\text{var } x: t; \text{vdecls}) \rho \ n_a \ st =_{\text{def}}$

- $\text{elab_vdecls } \text{vdecls } \rho[(n_a, \ st) / x] \ (n_a + \text{gr}(t)) \ st$ für nicht-Feld-Typen t

- $\text{elab_vdecls} (\text{var } x: \text{array } [u_1..o_1, \dots, u_k..o_k] \text{ of } t; \text{vdecls}) \rho \ n_a \ st =_{\text{def}}$

- $\text{elab_vdecls } \text{vdecls } \rho[(n_a, \ st) / x] \ (n_a + 3k + 2 + \prod_{i=1..k} (o_i - u_i + 1) \times \text{gr}(t)) \ st$

- falls x statisches Feld ist

Bei Parametern: 1 Speicherplatz für Adresse
Hier: Speicherplatz entsprechend Größe

Platz für
Deskriptor

Platz für
Feldkomponenten

Platzbedarf

- Variable: Speicherplatz entsprechender Größe
- (statisches) Feld: Platz für Deskriptor und Komponenten

Zusätzlich (zu lokalen Variablen)

- Verbundkomponenten: wie früher
- Dynamische Felder
 - Deskriptor in statischem Teil des Kellerrahmens
 - Bindung des Feldnamens an Anfangsadresse Deskriptor
 - Feld selbst im dynamischen Teil des Kellerrahmens abgelegt durch entsprechende Befehlsfolge bei Prozeduraufruf

Anders als früher

- Deskriptoren für alle Felder
(Deskriptorinformation wird benötigt, falls Feld aktueller Parameter einer Prozedur ist)
- Statische Felder
Deskriptor und Feld nacheinander (Eintragung durch entsprechende Befehlsfolge)

Verarbeitung lokaler Prozedurdeklarationen

- $\text{elab_pdecls}: \text{pdecl}^* \times \text{Adr_Umg} \times \text{ST} \rightarrow \text{Adr_Umg} \times \text{M}$ — *Folge von P-Befehlen*
- mit
 - $\text{elab_pdecls} () \rho \text{ st} =_{\text{def}} (\rho, ())$
 - $\text{elab_pdecls} (\mathbf{proc} p_1(\dots); \dots; \dots \mathbf{proc} p_k(\dots); \dots) \rho \text{ st} =_{\text{def}}$
 $(\rho', l_1: \text{code} (\mathbf{proc} p_1(\dots); \dots) \rho', \text{st}+1; \dots l_k: \text{code} (\mathbf{proc} p_k(\dots); \dots) \rho', \text{st}+1)$
wobei $\rho' = \rho[(l_1, \text{st}) / p_1, \dots, (l_k, \text{st}) / p_k]$
- Behandlung angewandter Vorkommen von Prozeduren vor definierendem Vorkommen durch Bindung der Prozedurnamen an symbolische Marken l_i (wegen möglicher Unterprogrammssprünge in p_i auf p_j mit $j > i$ und noch nicht bekanntem p_j)

Beachte

- In $\text{code} (\mathbf{proc} \dots)$ berechnete Adressumgebungen sind nach außen nicht bekannt



Prozedureintritt

- Situation
 - p sei momentan aktive Prozedur; alle Verweise seien richtig gesetzt; p rufe Prozedur q auf

Aktionen für Eintritt in q

- Durch aufrufende Prozedur p
 - SV-Verweis auf richtige Inkarnation der q direkt umfassenden Prozedur setzen
 - DV-Verweis auf Anfang Kellerrahmen von p setzen
 - Aktueller Stand EP-Register retten
 - Parameterübergabe (\rightarrow später: $code_A$);
 - MP-Register auf neuen Kellerrahmen setzen
 - Rücksprungadresse speichern
 - Sprung auf erste Instruktion der Übersetzung von q ausführen

} — **mit mst**

} — **mit cup**
- Durch aufgerufene Prozedur q
 - SP auf Anfang des lokalen Kellers von q setzen
 - Kopien der aktuellen value-Feldparameter anlegen (\rightarrow später: $code_S$)
 - EP setzen (einschließlich Kollision Keller - Halde prüfen)

} — **mit ssp**

} — **mit sep**

Prozedurverlassen

- Aktionen nach Abarbeitung von q
 - Restaurieren von MP und EP
 - Kellerrahmen von q freigeben
 - Rücksprung nach p

} — **mit retf bzw. retp**

Befehle für Prozedur-Eintritt

	Befehl	Bedeutung	Kommentar
mark stack	mst p	$STORE[SP + 2] := base(p, MP);$ $STORE[SP + 3] := MP;$ $STORE[SP + 4] := EP;$ $SP := SP + 5$	Verweis auf statischen Vorgänger Verweis auf dynamischen Vorgänger Retten von EP
call user procedure	cup p q	$MP := SP - (p + 4);$ $STORE[MP + 4] := PC;$ $PC := q$	p = Platzbedarf für die Parameter Retten der Rücksprungadresse Sprung zur Anfangsadresse q der Prozedur
set sp	ssp p	$SP := MP + p - 1;$	p = Größe statischer Teil des Datenbereichs
set ep	sep p	$EP := SP + p;$ if EP ≥ NP then error(„store overflow“) fi ;	p = maximale Tiefe des lokalen Kellers
$base(p,a) = \text{if } p = 0 \text{ then } a \text{ else } base(p-1, STORE[a+1]) \text{ fi}$			

Berechnet AA des Kellerrahmens des statischen Vorgängers

Befehle für Prozedur-Verlassen

	Befehl	Bedeutung	Kommentar
return function	retf	SP := MP; PC := STORE[MP + 4]; EP := STORE[MP + 3]; if EP ≥ NP then error(„store overflow“) fi; MP := STORE[MP + 2];	Funktionsergebnis oben im Keller Rücksprung Restaurieren von EP DV-Verweis
return procedure	retp	SP := MP - 1; PC := STORE[MP + 4]; EP := STORE[MP + 3]; if EP ≥ NP then error(„store overflow“) fi; MP := STORE[MP + 2];	Prozedur ohne Ergebnis Rücksprung Restaurieren von EP DV-Verweis

Aufgerufene Prozedur könnte NP verändert haben

Unterschied (zwischen Funktion und Prozedur)

- Funktion: SP auf 1. Rahmencelle der aufgerufenen Funktion (= Ergebnis)
- Prozedur: SP auf letzte besetzte Adresse des Aufrufers

Übersetzungsschema für Prozedurdeklaration

- code (**proc** p (specs); vdecls, pdecls; body) ρ st =_{def}

ssp n_a"; *Speicherbedarf statischer Teil (= Parameter + lokale Variable)*
code_S specs ρ' st; *Speicherbedarf dynamischer Teil (Feldkomponenten bei Feldparametern)*
code_P vdecls ρ'' st; *Speicherbedarf dynamischer Teil (Deskriptor für lokale Felder initialisieren)*
sep k; *Maximale Tiefe des lokalen Kellers*
ujp l;
proc_code; *Code für die lokalen Prozeduren*
l: code body ρ''' st; *Code für den Prozedurrumpf*
retp;

- wobei

- $(\rho', n_a') = \text{elab_specs specs } \rho \text{ } 5 \text{ st}$
- $(\rho'', n_a'') = \text{elab_vdecls vdecls } \rho' \text{ } n_a' \text{ st}$
- $(\rho''', \text{proc_code}) = \text{elab_pdecls pdecls } \rho'' \text{ st}$

- bei Funktionen

- **retf**-Befehl anstelle von **retp**-Befehl (hinterlässt Funktionsergebnis oben auf dem Keller)

Übersetzungsschema für Prozeduraufruf

- code $p(e_1, \dots, e_k) \rho \text{ st} =_{\text{def}}$

mst $\text{st} - \text{st}'$;

Differenz der Schachtelungstiefen für SV-Verweis

$\text{code}_A e_1 \rho \text{ st}; \dots$

$\text{code}_A e_k \rho \text{ st};$

}

Parameterübergabe (s.u.)

cup $s \mid$

- wobei

- s = Platzbedarf für die aktuellen Parameter

- $\rho(p) = (l, \text{st}')$

p zugeordnete Marke

p zugeordnete Schachtelungstiefe



Beispiel

- Geg.: $\rho = \{p \rightarrow (m, 1)\}$, $st = 2$

```
proc p (value n: integer; var x: integer);  
    var k: integer;    proc dupl (var x: integer); begin x := x × 2 end;  
    begin k := 0; while k < n do k := k+1; dupl(x) od end
```

- Ges.: code (**proc** p ...) ρ 2
 - $(\rho', n_a') = \text{elab_specs (value ...)} \rho$ 5 2 = $(\{p \rightarrow (m, 1), n \rightarrow (5, 2), x \rightarrow (6, 2)\}, 7)$
 - $(\rho'', n_a'') = \text{elab_vdecls (var k ...)} \rho'$ 7 2 = $(\{p \rightarrow (m, 1), n \rightarrow (5, 2), x \rightarrow (6, 2), k \rightarrow (7, 2)\}, 8)$
 - $(\rho''', \text{proc_code}) = \text{elab_pdecls (proc dupl ...)} \rho''$ 2 = $(\rho\rho, ll: \text{code (proc dupl ...)} \rho\rho$ 3)
wobei $\rho\rho = \{p \rightarrow (m, 1), n \rightarrow (5, 2), x \rightarrow (6, 2), k \rightarrow (7, 2)\}, \text{dupl} \rightarrow (ll, 2)\}$
- Nun ges.: code (**proc** dupl ...) $\rho\rho$ 3
 - $(\rho\rho', nn_a') = \text{elab_specs (var x ...)} \rho\rho$ 5 3 =
 $(\{p \rightarrow (m, 1), n \rightarrow (5, 2), x \rightarrow (5, 3), k \rightarrow (7, 2), \text{dupl} \rightarrow (ll, 2)\}, 6)$
 - $(\rho\rho'', nn_a'') = \text{elab_vdecls ()} \rho\rho'$ 6 3 = $(\rho\rho', 6)$
 - $(\rho\rho''', \text{proc_code}') = \text{elab_pdecls ()} \rho\rho''$ 3 = $(\rho\rho', ())$



Beispiel (Fortsetzung)

- Damit (für code (**proc** dupl ...) pp 3)

```

ssp 6;
sep 3;
ujp lll;
lll: lda 0 5; lda 0 5; ind i; ldc i 2; mult; sto i;
retp
    
```

Annotations:

- `ssp 6;` → Dynamischer Teil (fehlt hier)
- `ujp lll;` → Code für lokale Prozeduren (fehlt hier)
- `lll: lda 0 5; lda 0 5; ind i; ldc i 2; mult; sto i;` → Rumpf von dupl

- Insgesamt

```

ssp 8;
sep 3;
ujp l;
ll: «Code für dupl»;
l: lda 0 7; ldc i 0; sto i;
l': lda 0 7; ind i; lda 0 5; ind i; les;
fjp e;
lda 0 7; lda 0 7; ind i; ldc i 1; add; sto i;
mst 0; codeA x pp 2; cup 1 ll;
ujp l';
e: retp
    
```

Annotations:

- `ssp 8;` → Dynamischer Teil (fehlt hier)
- `ujp l;` → Wie oben
- `l: lda 0 7; ldc i 0; sto i;` → `k := 0`
- `l': lda 0 7; ind i; lda 0 5; ind i; les;` → `k < n`
- `lda 0 7; lda 0 7; ind i; ldc i 1; add; sto i;` → `k := k+1`
- `codeA x pp 2; cup 1 ll;` → `dupl(x)`
- The loop body (from `l'` to `cup`) is grouped as `while`.
- The entire loop structure is grouped as `Rumpf von p`.

Situation

- Verschiedene Arten von Parametern
 - var-Parameter
 - value-Parameter
 - Skalare
 - Verbunde + Felddeskriptoren (statisch)
 - Dynamische Felder
 - Prozeduren (formale Prozeduren)
- Kontext für Parameterübergabe
 - **mst**-Befehl ausgeführt (d.h. organisatorische Zellen besetzt)

Übersetzungsschema (für var-Parameter) ———— *Übergabe der Adresse („call-by-reference“)*

- $\text{code}_A \ x \ \rho \ \text{st} =_{\text{def}} \text{code}_L \ x \ \rho \ \text{st}$ ———— *code_L unten neu definiert*
falls der zu x korrespondierende formale Parameter **var**-Parameter ist

Übersetzungsschema (für skalare value-Parameter) ———— *Übergabe des Werts („call-by-value“)*

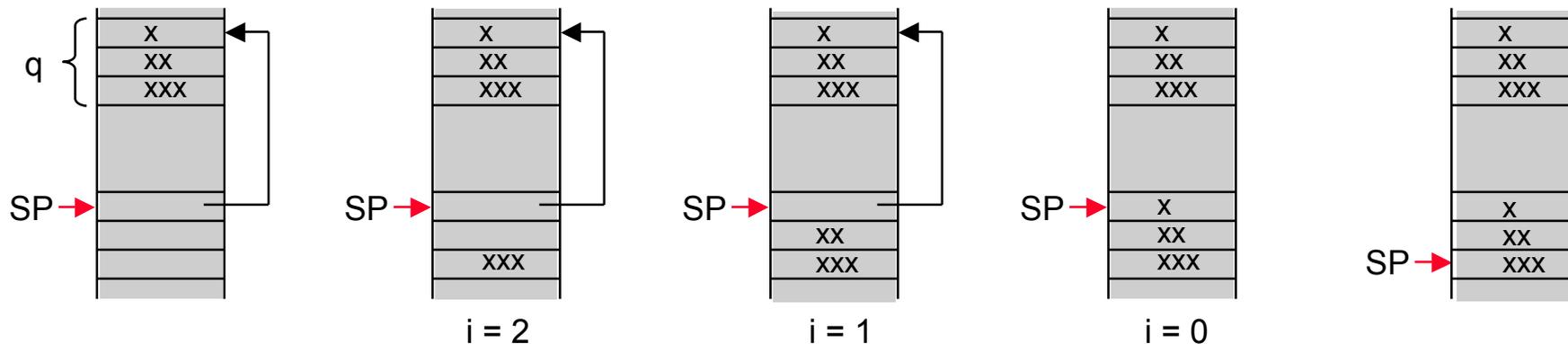
- $\text{code}_A \ e \ \rho \ \text{st} =_{\text{def}} \text{code}_R \ e \ \rho \ \text{st}$
falls der zu e korrespondierende formale Parameter **value**-Parameter ist

Übersetzungsschema (für value-Parameter – Verbunde + Deskriptoren von Feldern)

- $\text{code}_A \ x \ \rho \ \text{st} =_{\text{def}} \text{code}_L \ x \ \rho \ \text{st}; \text{ movs } g$
 falls der zu x korrespondierende formale **value**-Parameter strukturierten Typs t und statischer Größe $gr(t) = g$ ist

*AA übergeben
Parameter kopieren*

Befehl	Bedeutung	Bedingung	Ergebnis
movs q	for $i := q - 1$ downto 0 do STORE[SP + i] := STORE[STORE[SP] + i] od ; SP := SP + $q - 1$	(a)	



Aktionen bei der Übersetzung von (dynamischen) **value-Feldparametern**

- Durch den Aufrufer
 - Deskriptor des aktuellen Feldparameters in Parameterbereich der aufgerufenen Prozedur kopieren (mit **movs**)
- Durch die aufgerufene Prozedur
 - Feldinhalt kopieren (mit **movd**)
 - Eintragen der fiktiven Anfangsadresse der Kopie in den Deskriptor

Übersetzungsschema (für die Spezifikation eines formalen value-Feldparameters x)

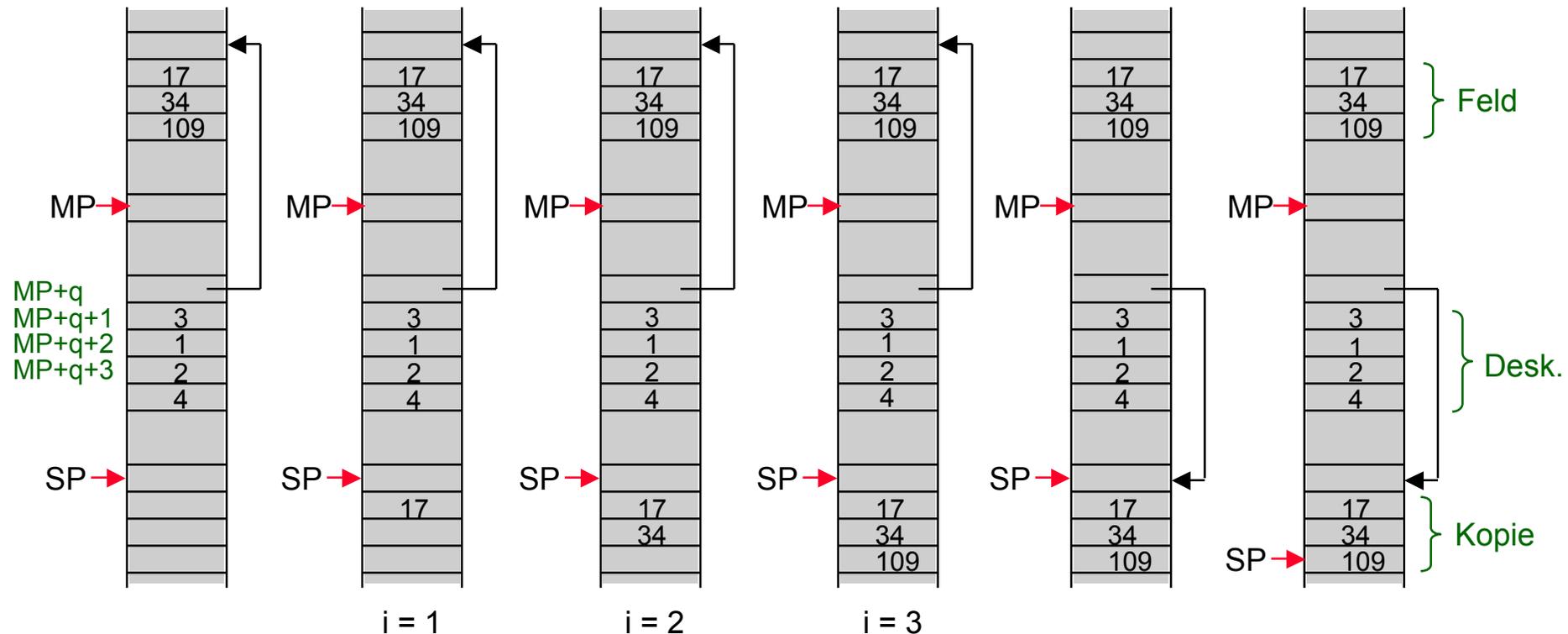
- Situation
 - Aufrufer hat Deskriptor des aktuellen Parameters bereits kopiert ($\rho(x) = (ra, st)$)
- Codefunktion zur Ablage der Feldkomponenten im dynamischen Teil (die bei Übersetzung der Prozedurdeklaration aufgerufen wird)
 - $code_S(\mathbf{value\ } x: \mathbf{array\ } [u_1..o_1, \dots, u_k..o_k] \mathbf{of\ } t; sp) \rho\ st =_{def} \mathbf{movd\ } ra; code_S\ sp\ \rho\ st$
 - $code_S(s; sp) \rho\ st =_{def} code_S\ sp\ \rho\ st$ falls s kein formaler **value**-Feldparameter ist
 - $code_S() \rho\ st =_{def} ()$

Kopieren eines dynamischen Feldes

Im Prinzip wie statisches Kopieren
(indirekt über Felddeskriptor)

Befehl	Bedeutung
movd q	for i:= 1 to STORE[MP + q + 1] do
	STORE[SP + i] := STORE[STORE[MP + q] + STORE[MP + q + 2] + i - 1] od ;
	STORE[MP + q] := SP + 1 - STORE[MP + q + 2];
	SP := SP + STORE[MP + q + 1];

q: Relativdr. Desk.
STORE[MP+q+1]:
Feldgröße
STORE[MP+q]: fikt. AA
STORE[MP+q+2]:
Subtr. für fikt. AA



Erweiterung von code_L (für Zugriff auf lokale/globale Variablen und formale Parameter)

- Lokale Variablen und formale **value**-Parameter

direkt: Relativadresse zu MP addieren

- Globale Variablen

direkt: Basisadresse mittels SV-Verweis berechnen und Relativadresse addieren

- Formale **var**-Parameter

indirekt: Relativadresse zu MP addieren

Spezialfall von globalen Variablen:
Basisadresse = MP

Übersetzungsschema code_L (für angewandte Vorkommen von Namen)

- $\text{code}_L(x\ r)\ \rho\ st =_{\text{def}} \mathbf{lda}\ d\ ra; \text{code}_M\ r\ \rho\ st$ ——— **lda: Adresse direkt**
wobei $\rho(x) = (ra, st')$ und $d = st - st'$, falls x Variable oder formaler **value**-Parameter ist

- $\text{code}_L(x\ r)\ \rho\ st =_{\text{def}} \mathbf{lod}\ d\ ra; \text{code}_M\ r\ \rho\ st$ ——— **lod: Adresse indirekt**
wobei $\rho(x) = (ra, st')$ und $d = st - st'$, falls x formaler **var**-Parameter ist

Motivation

- Geg.: Beispiel
- Ges.: SV-Verweis für das Anlegen des Kellerrahmens für die aktuellen Prozeduren (d.h. für g bzw. f bei Aufruf von p)

Wegen statischer Bindung

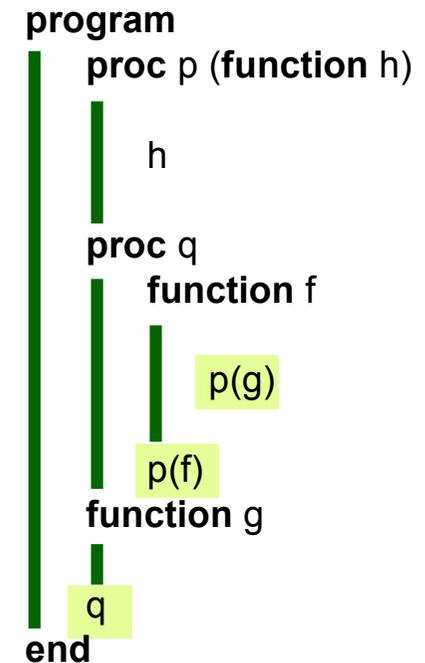
- Für Aufrufer von p muss aktuelle Prozedur sichtbar sein (d.h. über Kette der statischen Vorgänger erreichbar)

Daher

- Kette der statischen Vorgänger liefert Anfangsadresse der richtigen Inkarnation
- Diese Adresse wird beim Aufruf mitgegeben

Übergabe einer aktuellen Prozedur

- Im Parameterbereich der aufgerufenen Prozedur werden abgelegt
 - Anfangsadresse der Übersetzung der aktuellen Prozedur
 - Anfangsadresse des Kellerrahmens ihres statischen Vorgängers



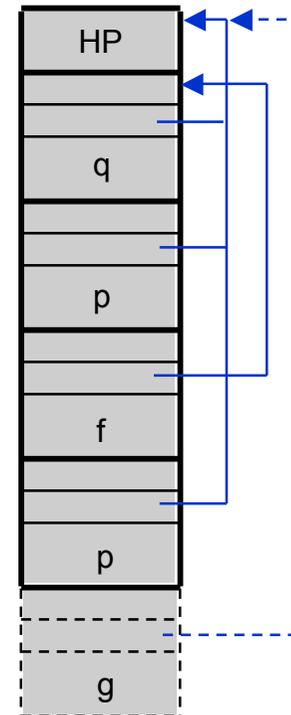
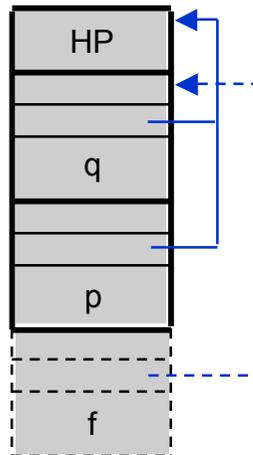
in Prozedurdeskriptor



Beispiel (SV-Verweis für formale Prozeduren)

```

program
  proc p (function h)
    h
  proc q
    function f
      p(g)
      p(f)
    function g
  q
end
  
```



Zwei mögliche Fälle (bei der Übergabe einer aktuellen Prozedur q in Aufruf p(q))

- Unterschied: Deskriptor existiert bereits oder muss erst angelegt werden
- Aktuelle Prozedur q ist deklariert: Deskriptor muss angelegt werden
 - „Prozedurdeskriptor“ (mit Anfangsadresse der Übersetzung der Prozedur und SV-Verweis) anlegen (mit p als Typ für Adresse in CODE)
 - $code_A f \rho st =_{def}$ falls f deklarierte Prozedur mit $\rho(f) = (adr, st')$ und $d = st - st'$ ist

ldc p adr;	Anfangsadresse der Übersetzung	}		<i>Deskriptor</i>
lda d 0;	SV-Verweis für späteren Aufruf			
- Aktuelle Prozedur q ist formale Prozedur (d.h. Parameter): Deskriptor existiert bereits
 - Inhalt des Deskriptors der formalen Prozedur kopieren
 - $code_A f \rho st =_{def}$ falls f formale Prozedur mit $\rho(f) = (ra, st')$ und $d = st - st'$ ist

lda d ra;	Lade Deskriptoradresse (ra ist Relativadresse des Deskriptors)
movs 2;	Kopiere Deskriptor

Aufruf einer formalen Prozedur

- Änderungen (gegenüber Aufruf einer deklarierten Prozedur)
 - SV-Verweis aus 2. Zelle der formalen Prozedur (modifizierter **mst**-Befehl)
 - Unterprogramm sprung indirekt über 1. Zelle der formalen Prozedur (neuer Befehl **cupi**)
- Übersetzungsschema für Aufruf einer formalen Prozedur f
 - $\text{code } f(e_1, \dots, e_k) \rho \text{ st} =_{\text{def}} \text{mstf st-st' ra}; \text{code}_A e_1 \rho \text{ st}; \dots \text{code}_A e_k \rho \text{ st}; \text{smp s}; \text{cupi (st-st')} \text{ra}$
 - wobei $\rho(f) = (\text{ra}, \text{st}')$ ist und s der Platzbedarf für die aktuellen Parameter

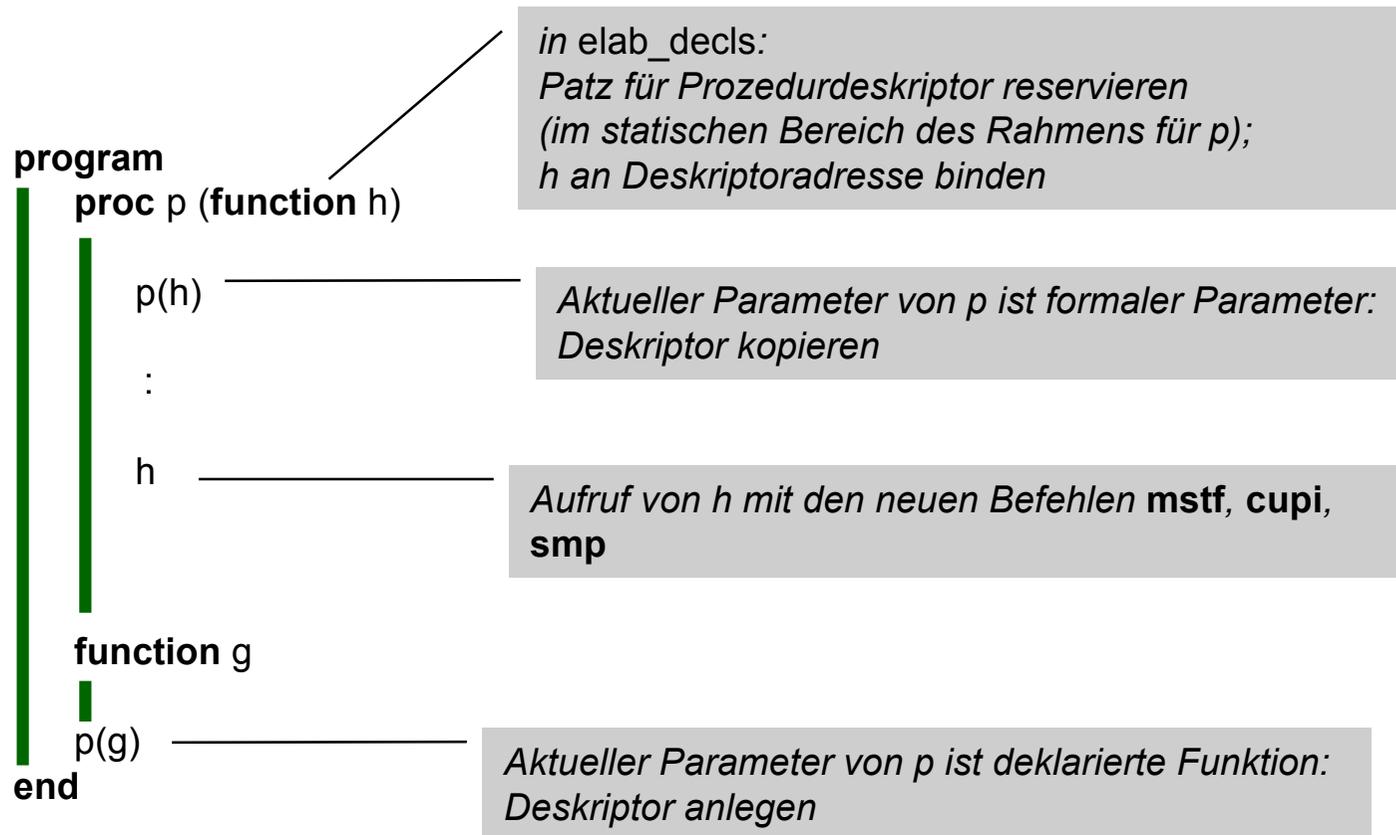
Befehl	Bedeutung	Kommentar
mstf p q	$\text{STORE}[\text{SP} + 2] := \text{STORE}[\text{base}(p, \text{MP}) + q + 1];$ $\text{STORE}[\text{SP} + 3] := \text{MP};$ $\text{STORE}[\text{SP} + 4] := \text{EP};$ $\text{SP} := \text{SP} + 5$	Verweis auf statischen Vorgänger Verweis auf dynamischen Vorgänger Retten von EP
cupi p q	$\text{STORE}[\text{MP} + 4] := \text{PC};$ $\text{PC} := \text{STORE}[\text{base}(p, \text{STORE}[\text{MP} + 2])] + q]$	Retten der Rücksprungadresse Sprung zur Anfangsadresse q der Prozedur
smp p	$\text{MP} := \text{SP} - (p + 4);$	

Erweiterung von `elab_specs` um formale Prozeduren und Funktionen

- Namen an Relativadressen für zugehörige Deskriptoren binden



Verschiedene Aktivitäten bei der Übersetzung formaler Prozeduren



Ausgangssituation (Initialisierung der Register)

- SP-Register mit -1 initialisiert
- NP-Register mit maxstore+1 initialisiert
- Übrige Register mit 0 initialisiert

Übersetzungsschema für das Hauptprogramm

- code (program vdecls, pdecls; stats) $\emptyset 0 =_{\text{def}}$

ssp n_a;

code_p vdecls ρ 1;

sep k;

ujp l;

proc_code;

l: code stats ρ' 1;

stp;

- wobei

■ (ρ, n_a) = elab_vdecls vdecls $\emptyset 5 1$

■ (ρ', proc_code) = elab_pdecls pdecls ρ 1

*Im Prinzip wie Prozeduren;
Unterschiede:
- feste Startwerte für Speichervergabe
- keine Parameter*

*Erzeugt Code für die
Besetzung von Felddeskriptoren*

hält die P-Maschine an